An Efficient Algorithm for δ-Approximate Matching with α-Bounded Gaps in Musical Sequences

Domenico Cantone, Salvatore Cristofaro, and Simone Faro

Università di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125 Catania, Italy
{cantone | cristofaro | faro}@dmi.unict.it

Abstract. We present a new efficient algorithm for the δ-approximate matching problem with α-bounded gaps. The δ-approximate matching problem, recently introduced in connection with applications in music retrieval, generalizes the exact string matching problem by relaxing the notion of matching. Here we consider the case in which matchings may contain bounded gaps.
An extensive comparison with the only (to our knowledge) other solution existing in literature for the same problem, due to Crochemore *et al.*, indicates that our algorithm is more efficient, especially in the cases of large alphabets and long patterns. In addition, our algorithm computes the total number of approximate matchings for each position of the text, requiring only $\mathcal{O}(m\alpha)$-space, where m is the length of the pattern.
Key words: approximate string matching, experimental algorithms, musical information retrieval.

1 Introduction

Given a text T and a pattern P over some alphabet Σ, the *string matching problem* consists in finding *all* occurrences of P in T. It is a very extensively studied problem in computer science, mainly due to its direct applications to such diverse areas as text, image and signal processing, speech analysis and recognition, information retrieval, computational biology and chemistry, etc.

Recently, the classical string matching problem has been generalized with various notions of approximate matching, particularly useful in specific fields such as molecular biology [KMGL88], musical applications [CIR98], or image processing [KPR00].

In this paper we focus on a variant of the approximate string matching problem, namely the *δ-approximate string matching problem with α-bounded gaps*. Such a problem, which will be given a precise definition later, arises in many questions concerning musical information retrieval and musical analysis. This is especially true in the context of monophonic music, in which one wants to retrieve a given melody from a complex musical score. We mention here that a significant amount of research has been devoted to adapt solutions for exact string matching to δ-approximate matching (see for instance [CCI+99], [CILP01], [CIL+02],

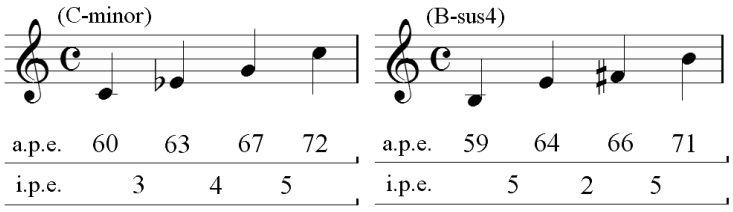

Fig. 1. Representation of the C-minor and B-sus4 chords in the absolute pitch encoding (a.p.e.) and in the interval pitch encoding (i.p.e.).

[CCF04]). In this respect, Boyer-Moore-type algorithms are of particular interest, since they are very fast in pratice.

The paper is organized as follows. In Section 2 we discuss the applications of approximate matching in the context of musical sequences. Then in Section 3 we introduce some basic notions and give a formal definition of the δ-approximate matching problem with α-bounded gaps. An algorithm based on the dynamic programming approach for the approximate matching problem of our interest is reviewed in Section 4. Then, in Section 5, we present a new efficient algorithm for the same problem. Experimental data obtained by running under various conditions both our algorithm and the one based on the dynamic programming approach are presented and compared in Section 6. Finally, we draw our conclusions in Section 7.

2 Approximate matching in musical sequences

Musical sequences can be schematically viewed as sequences of integer numbers, representing either the notes in the chromatic or diatonic notation (absolute pitch encoding), or the intervals, in number of semitones, between consecutive notes (interval pitch encoding); see the examples in Fig. 1. The second representation is generally of greater interest for applications in tonal music, since absolute pitch encoding disregards tonal qualities of pitches. Note durations and note accents can also be encoded in numeric form, giving rise to richer alphabets whose symbols can really be regarded as sets of parameters. This is the reason why alphabets used for music representation are generally quite large.

δ-approximate string matching algorithms are very effective to search for all similar but not necessarily identical occurrences of given melodies in musical scores. We recall that in the δ-approximate matching problem two integer strings of the same length match if the corresponding integers differ by at most a fixed bound δ. For instance, the chords C-minor and B-sus4 match if a tolerance of $\delta = 1$ is allowed in the absolute pitch encoding (where C-minor$= (60, 63, 67, 72)$ and B-sus4$= (59, 64, 66, 71)$), whereas if we use the interval pitch encoding, a tolerance of $\delta = 2$ is required to get a match (in this case we have C-minor$= (3, 4, 5)$

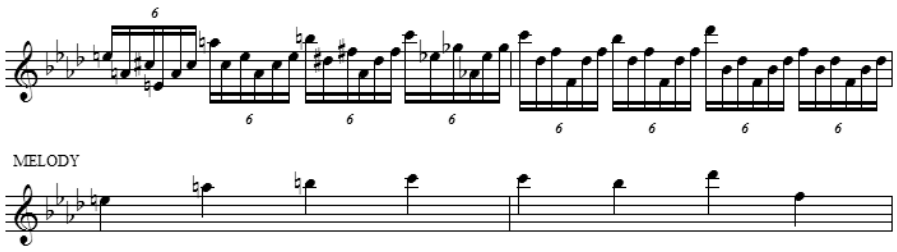

Fig. 2. Two bars of the study Op. 25 N. 1 of F. Chopin (first score). The second score represents the melody.

and B-sus4= $(5, 2, 5)$); see Fig. 1. Notice that for $\delta = 0$, the δ-approximate string matching problem reduces to the exact string matching problem.

Intuitively, we say that a melody (or *pattern*) has a δ-approximate occurrence with α-bounded gaps within a given musical score (or *text*), if the melody has a δ-approximate matching with a subsequence of the musical score, in which it is allowed to skip up to a fixed number α of symbols (the *gap*) between any two consecutive positions. In the present context, two symbols have an approximate matching if the absolute value of their difference is bounded by a fixed number δ.

In classical music compositions, and in particular in compositions for *Piano Solo*, it is quite common to find musical pieces based on a sweet ground melody, whose notes are interspaced by rapidly executed arpeggios. Fig. 2 shows two bars of the study *Op. 25 N. 1 for Piano Solo* by F. Chopin illustrating such a point. The notes of the melody are the first of each group of six notes (sextuplet). If we use the standard MIDI representation of the pitches, then the melody corresponds to the sequence of integer numbers $P = [76, 81, 83, 84, 84, 83, 86, 77]$. Then, if a gap bound of $\alpha = 5$ is allowed, an exact occurrence of the melody can be found through the piece.

The above musical technicality is not by any means the only one for which approximate string matching with bounded gaps turns out to be very useful. Other examples are given by *musical ornaments*, which are common practice in classical music, and especially in the music of the baroque period. Musical ornaments are groups of notes, played at a very fast tempo, which generally are "attached" to the notes of a given melody, in order to emphasize or adorn certain dynamical passages. Some of the most common musical ornaments are the *acciaccatura*, the *appoggiatura*, the *mordent*, the *turn*, and the *trill*.

Fig. 3 shows an excerpt of a *Minuet* by J.S. Bach, which makes use of musical ornaments. We provide both the actual score, in which ornaments are represented as special symbols marked above notes or as groups of small notes, and the corresponding score showing how these notations translate into real musical execution. Note that in Fig. 3 the mordent corresponds to a group of three notes, whereas the trill corresponds to a group of 16 notes. In general, to take care of musical ornaments in δ-matching problem with gaps, one needs gap values in the range between 4 and 16.

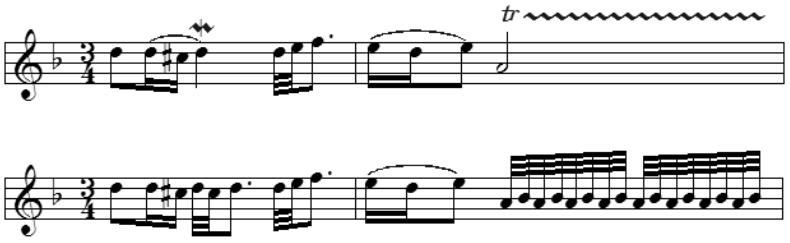

Fig. 3. An excerpt of a piece of J.S. Bach (first score). The second score shows how the musical ornaments must be played. Two musical ornaments are present: a *mordent*, attached to the 4^{th} note, and a *trill*, attached to the 11^{th} note.

3 Basic definitions and properties

Before entering into details, we need a bit of notations and terminology. A string P is represented as a finite array $P[0\,..\,m-1]$, with $m \geq 0$. In such a case we say that P has length m and write $\text{length}(P) = m$. In particular, for $m = 0$ we obtain the empty string. By $P[i]$ we denote the $(i+1)$-st character of P, for $0 \leq i < \text{length}(P)$. Likewise, by $P[i\,..\,j]$ we denote the substring of P contained between the $(i+1)$-st and the $(j+1)$-st characters of P, for $0 \leq i \leq j < \text{length}(P)$. The substrings of the form $P[0\,..\,j]$ (also denoted by P_j), with $0 \leq j < \text{length}(P)$, are the nonempty *prefixes* of P.

Let Σ be an alphabet of integer numbers and let $\delta \geq 0$ be an integer. Two symbols a and b of Σ are said to be *δ-approximate* (or we say that a and b *δ-match*), in which case we write $a =_\delta b$, if $|a-b| \leq \delta$. Two strings P and Q over the alphabet Σ are said to be δ-approximate (or we say that P and Q δ-match), in which case we write $P \stackrel{\delta}{=} Q$, if

$$\text{length}(P) = \text{length}(Q), \quad \text{and } P[i] =_\delta Q[i], \quad \text{for } i = 0, ..., \text{length}(P) - 1\,.$$

Given a text T of length n and a pattern P of length m, a *δ-occurrence with α-bounded gaps of P in T at position i* is an increasing sequence of indices $(i_0, i_1, \ldots, i_{m-1})$ such that (i) $0 \leq i_0$ and $i_{m-1} = i \leq n-1$, (ii) $i_{h+1} - i_h \leq \alpha + 1$, for $h = 0, 1, \ldots m-2$, and (iii) $P[j] =_\delta T[i_j]$, for $j = 0, 1, \ldots m-1$. We write $P \trianglelefteq_{\delta,\,\alpha} T_i$ to mean that P has a δ-occurrence with α-bounded gaps in T at position i (in fact, when the bounds δ and α are well understood from the context, we will simply write $P \trianglelefteq T_i$).

The *δ-approximate string matching problem with α-bounded gaps* admits the following variants: (a) find all δ-occurrences with α-bounded gaps of P in T; (b) find all positions i in T such that $P \trianglelefteq T_i$; (c) for each position i in T, find the number of distinct δ-occurrences of P with α-bounded gaps at position i.

In Section 5 we will describe an efficient $\mathcal{O}(mn)$-time solution for the variants (b) and (c) above which uses only $\mathcal{O}(m\alpha)$ extra space. Variant (a) can then be

solved by running an $\mathcal{O}(m^2\alpha)$-time and -space local search at each position i such that $P \trianglelefteq T_i$.

The following very elementary fact will be used later.

Lemma 1. *Let T and P be a text of length n and a pattern of length m, respectively. Also, let $\delta, \alpha \geq 0$. Then, for each $0 \leq i < n$ and $0 \leq k < m$, we have that $P_k \trianglelefteq_{\delta,\,\alpha} T_i$ if and only if $P[k] =_\delta T[i]$ and either $k = 0$, or $P_{k-1} \trianglelefteq_{\delta,\,\alpha} T_{i-h}$, for some h such that $1 \leq h \leq \alpha + 1$.* ■

4 A dynamic programming algorithm: δ-Bounded-Gaps

The δ-approximate matching problem with α-bounded gaps has been first addressed by Crochemore *et al.* in [**?**], where an algorithm based on the dynamic programming approach —named δ-Bounded-Gaps— has been proposed. To our knowledge, this is still the only solution present in literature for the δ-approximate matching problem with α-bounded gaps. In our review, we follow the presentation given later in [CIM$^+$02], which considers also several new versions of the approximate matching problem with gaps.

Given as usual a text T of length n, a pattern P of length m, and two integers $\delta, \alpha \geq 0$, the algorithm δ-Bounded-Gaps runs in $\mathcal{O}(mn)$-time and -space, at least in the case in which one is interested in finding all δ-occurrences with α-bounded gaps of P in T (variant (a)). Space requirements can be reduced to $\mathcal{O}(n)$, if only positions i in T such that $P \trianglelefteq T_i$ need to be computed (variant (b)). To solve also variant (c) with δ-Bounded-Gaps, one needs to first solve variant (a) and then trace back and count all approximate matchings with gaps at each position of the text T.

The algorithm δ-Bounded-Gaps is presented as an incremental procedure, based on the dynamic programming approach. More specifically, it starts by first computing all the δ-occurrences with α-bounded gaps in T of the prefix of P of length 1, i.e. P_0. Then, during the i-th iteration, it looks for all the δ-occurrences with α-bounded gaps in T of the prefix P_{i-1}. At the end of the last iteration, the δ-occurrences of the whole pattern P have been computed.

To give a more formal description of the algorithm, let us put:

$$\mathit{LastOccur}_j(P_i) = \max\left(\{0 \leq k \leq j : P_i \trianglelefteq T_k \text{ and } j - k \leq \alpha\} \cup \{-1\}\right) .$$

Notice that if $\mathit{LastOccur}_j(P_i) = -1$, then $P_i \not\trianglelefteq T_k$ for $k = j - \alpha, j - \alpha + 1, \ldots, j$. Otherwise, $\mathit{LastOccur}_j(P_i)$ is the largest value $k \in \{j - \alpha, j - \alpha + 1, \ldots, j\}$ such that $P_i \trianglelefteq T_k$.

Using a *trace-back* procedure, as described in [CIM$^+$02], the values $\mathit{LastOccur}_j(P_i)$ can be used to retrieve the approximate matchings at a given position in time $\mathcal{O}(m\alpha)$.

The values $\mathit{LastOccur}_j(P_i)$ can be computed incrementally, for $i = 0, 1, \ldots, m-1$ and $j = 0, 1, \ldots, n-1$. More specifically, the algorithm δ-Bounded-Gaps fills a matrix D of dimension $m \times n$, where $D[i, j]$ corresponds to $\mathit{LastOccur}_j(P_i)$, according to the following recursive relation:

```
δ-Bounded-Gaps (P, T, δ, α)
 1.   n = length(T)
 2.   m = length(P)
 3.   for i = 0 to m − 1 do
 4.        D[i, 0] = −1
 5.   for i = 0 to m − 2 do
 6.        for j = 1 to n − 1 do
 7.             D[i, j] = −1
 8.             if P[i] =_δ T[j] then
 9.                  if i = 0 or D[i − 1, j − 1] ≥ 0  then
10.                       D[i, j] = j
11.             else if D[i, j − 1] ≥ j − α then
12.                  D[i, j] = D[i, j − 1]
13.   for j = m − 1 to n − 1 do
14.        if P[m − 1] =_δ T[j] and D[m − 2, j − 1] ≥ 0 then
15.             output(j)
```

Fig. 4. The algorithm δ-Bounded-Gaps for the δ-approximate matching problem with α-bounded gaps.

$$D[i,j] = \begin{cases} j & \text{if } T[j] =_\delta P[i]\,,\ i,j \geq 1\,,\ \text{and } D[i-1,j-1] \geq 0 \\ j & \text{if } T[j] =_\delta P[i] \text{ and } i = 0 \\ D[i,j-1] & \text{if } T[j] \neq_\delta P[i]\,,\ j \geq 1\,,\ \text{and } D[i,j-1] \geq j - \alpha \\ -1 & \text{otherwise} \end{cases}$$

where $0 \leq i < m$ and $0 \leq j < n$.

Fig. 4 presents the pseudo-code of the algorithm δ-Bounded-Gaps. Its running time is easily seen to be $\mathcal{O}(mn)$. Also, $\mathcal{O}(mn)$-space is needed to store the matrix D. However, if one is only interested in the positions i of T at which $P \trianglelefteq T_i$, space requirements reduce to $\mathcal{O}(n)$, since the computation of each row depends only on the values stored in the previous row.

5 A new efficient algorithm: (δ, α)-Sequential-Sampling

In this section we present a new efficient algorithm for the δ-approximate matching problem with α-bounded gaps, named (δ, α)-Sequential-Sampling. Our algorithm is characterized by an $\mathcal{O}(mn)$-time and an $\mathcal{O}(m\alpha)$-space complexity, where m and n are the length of the pattern and text, respectively. Notice that in practical applications $m\alpha$ is much smaller than n. In addition, our algorithm solves variant (c) (and therefore also variant (b)) of the approximate matching problem with gaps, as stated in Section 3, namely it computes the number of distinct δ-occurrences of the patter with α-bounded gaps at each position of the text. If

one is also interested in retrieving the actual approximate matching occurrences at position i of a text T, a possibility would be to compute the submatrix $D[k, j]$, for $\max(0, (m-1)\cdot(\alpha+1)) \leq k \leq i$ and $0 \leq j \leq m-1$, where, as before, m is the length of the pattern, and then trace back through all possible approximate matchings. The submatrix $D[k, j]$ can be computed in time and space $\mathcal{O}(m^2\alpha)$ by the algorithm δ-Bounded-Gaps.

Our algorithm follows a different computation ordering than the one followed by the algorithm δ-Bounded-Gaps; in fact, it computes the occurrences of all prefixes of the pattern in continuously increasing prefixes of the text, rather than computing all occurrences in the whole text of continuously increasing prefixes of the pattern, as the algorithm δ-Bounded-Gaps does. That is, for a text T of length n, pattern P of length m, and nonnegative integers δ, α, during its first iteration the algorithm (δ, α)-Sequential-Sampling computes the (number of) occurrences of all prefixes P_k of P such that $P_k \trianglelefteq T_0$. Then, during the i-th iteration, it computes (the number of) all occurrences of prefixes P_k of P such that $P_k \trianglelefteq T_i$, using information gathered during previous iterations.

To be more precise, let $\mathcal{S}_i$ denote the collection of all pairs (j, k) such that $P_k \trianglelefteq T_j$, for $0 \leq i \leq n$, $0 \leq j < i$, and $0 \leq k < m$. Notice that $\mathcal{S}_0 = \varnothing$. If we put $\mathcal{S} = \mathcal{S}_n$, then the problem of finding the positions i in T such that $P \trianglelefteq T_i$ translates to the problem of finding all values i such that $(i, m-1) \in \mathcal{S}$.

To begin with, notice that Lemma 1 justifies the following recursive definition of the set $\mathcal{S}_{i+1}$ in terms of $\mathcal{S}_i$, for $i < n$:

$$\mathcal{S}_{i+1} = \mathcal{S}_i \cup \{(i,k) : P[k] =_\delta T[i] \text{ and } (k = 0 \text{ or } (i-h, k-1) \in \mathcal{S}_i, \text{ for some } h \in \{1, \ldots, \alpha+1\})\}.$$

Such relation, coupled with the initial condition $\mathcal{S}_0 = \varnothing$, allows one to compute the set $\mathcal{S}$ in an iterative fashion, as shown in Fig. 5. The time complexity of the resulting algorithm —named Slow-(δ, α)-Sequential-Sampling— is $\mathcal{O}(nm\alpha)$. Notice that given the set $\mathcal{S}$, one can easily compute the actual δ-occurrences with α-gaps of P in T.

From a practical point of view, the set $\mathcal{S}$ in the algorithm Slow-(δ, α)-Sequential-Sampling could be represented by its characteristic $n \times m$ matrix $\mathcal{M}$, where $\mathcal{M}[i, k]$ is 1 or 0, provided that the pair (i, k) belongs or does not belong to $\mathcal{S}$, for $0 \leq i < n$ and $0 \leq k < m$.

Since during the i-th iteration of the **for**-loop at line 4 of the algorithm Slow-(δ, α)-Sequential-Sampling at most $\alpha + 1$ rows of $\mathcal{M}$ need to be scanned —more precisely the ones having index $j \in \{\max(0, i-\alpha-1), i-1\}$,— it would be enough to store only $\alpha + 1$ rows of $\mathcal{M}$ at each step of the computation, plus another one as working area.

In addition, by maintaining an extra array $\mathcal{C}$ of length m such that during the i-th iteration of the **for**-loop at line 4 the following invariant holds:

$$\mathcal{C}[k] = \sum_{j=\max(0,i-\alpha-1)}^{i-1} \mathcal{M}[j, k]\,, \qquad \text{for } 0 \leq k < m\,,$$

```
Slow-(δ, α)-Sequential-Sampling (T, P, δ, α)
 1.    n = length(T)
 2.    m = length(P)
 3.    S = ∅
 4.    for i = 0 to n − 1 do
 5.        for k = m − 1 downto 1 do
 6.            if P[k] =_δ T[i] and (i − h, k − 1) ∈ S, for some h ∈ {1, ..., α + 1} then
 7.                S = S ∪ {(i, k)}
 8.        if P[0] =_δ T[i] then
 9.            S = S ∪ {(i, 0)}
10.    for i = 0 to n − 1 do
11.        if (i, m − 1) ∈ S then
12.            output(i)
```

Fig. 5. The algorithm Slow-(δ, α)-Sequential-Sampling for the δ-approximate matching problem with α-bounded gaps.

the test of the conditional instruction at line 6 can be performed in constant time, rather than in $\mathcal{O}(\alpha)$-time.

Such observations allow to reduce the space requirement to $\mathcal{O}(m\alpha)$ and the running time to $\mathcal{O}(mn)$.

In fact, we can do a little bit more than that. Rather than maintaining in $\mathcal{M}[j,k]$ the Boolean value of the test $(j,k) \in \mathcal{S}$, it is more convenient to let $\mathcal{M}[j,k]$ count the number of *distinct* δ-occurrences with α-gaps of P_k at position j of T. With this change, when the i-th iteration of the **for**-loop at line 4 of algorithm Slow-(δ, α)-Sequential-Sampling starts, the item $\mathcal{C}[k]$ will contain the total number of *distinct* δ-occurrences with α-gaps of P_k at positions $\max(0, i - \alpha - 1)$ through $i - 1$, provided that the above invariant holds. Such values can then be used to maintain the invariant itself.

Plainly, at the end of the computation one can retrieve in constant time the number of approximate matchings at each position of the text.

The resulting algorithm —named (δ, α)-Sequential-Sampling— is presented in detail in Fig. 6. Its overall running time is $\mathcal{O}(mn)$ and its space requirement is $\mathcal{O}(m\alpha)$.

6 Experimental results

In this section we report experimental data relative to an extensive comparison of the running times of our algorithm (δ, α)-Sequential-Sampling, presented in Section 5, and the algorithm δ-Bounded-Gaps, described in Section 4.

Both algorithms have been implemented in the C programming language and were used to search for the same patterns in large fixed text sequences on a PC with a Pentium IV processor at 2.66GHz. In particular, they have been tested on three Randσ problems, for $\sigma = 60, 120, 180$ and on a real music text buffer.

```
(δ, α)-Sequential-Sampling (T, P, δ, α)
 1.    n = length(T)
 2.    m = length(P)
 3.    for i = 0 to α + 1 do
 4.        for j = 0 to m − 2 do
 5.            M[i, j] = 0
 6.    for i = 0 to m − 2 do C[i] = 0
 7.    for i = 0 to n − 1 do
 8.        j = i mod(α + 2)
 9.        for k = 0 to m − 2 do
10.            C[k] = C[k] − M[j, k]
11.            M[j, k] = 0;
12.        if P[m − 1] =δ T[i] and C[m − 2] > 0 then
13.            output(i)
14.        for k = m − 2 downto 1 do
15.            if P[k] =δ T[i] and C[k − 1] > 0 then
16.                M[j, k] = C[k − 1]
17.                C[k] = C[k] + C[k − 1]
18.        if P[0] =δ T[i] then
19.            M[j, 0] = 1
20.            C[0] = C[0] + 1
```

Fig. 6. The (δ, α)-Sequential-Sampling algorithm for the δ-approximate matching problem with α-bounded gaps.

In particular, each Randσ problem consisted in searching a set of 250 random patterns of length 10, 20, 40, 60, 80, 100, 120, and 140 in a 5Mb random text sequence over a common alphabet of size σ. For each Randσ problem, the approximation bound δ and the gap bound α have been set to 1, 2, 4 and to 4, 8, respectively.

Concerning the tests on the real music text buffer, these have been performed on a 4.8Mb file obtained by combining a set of classical pieces, in MIDI format, by C. Debussy. The resulting text buffer has been translated in the pitch interval encoding with an alphabet of 101 symbols. For each $m = 10, 20, 40, 60, 80, 100, 120, 140$, we have randomly selected in the file 250 substrings of length m which subsequently have been searched for in the same file.

All running times in the tables have been expressed in tenths of second and, for each length of the pattern, the best result achieved has been bold-faced. Moreover, (δ, α)-S-S denotes our algorithm (δ, α)-Sequential-Sampling, whereas δ-B-G denotes the algorithm δ-Bounded-Gaps by Crochemore *et al.*

Experimental results with $\sigma = 60$

$\delta = 1, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.042**	**8.203**	**14.26**	**20.41**	**26.38**	**32.36**	**38.43**	**44.47**
δ-B-G	5.724	10.75	21.13	32.03	42.80	53.84	64.87	75.77
$\delta = 1, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.199**	**8.511**	**14.56**	**20.69**	**26.57**	**32.53**	**38.80**	**44.85**
δ-B-G	5.660	10.52	21.28	31.69	42.55	53.19	64.20	74.36
$\delta = 2, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.336**	**8.580**	**14.73**	**20.86**	**26.72**	**32.73**	**38.84**	**44.85**
δ-B-G	5.832	11.00	22.67	33.34	44.80	55.99	67.29	78.56
$\delta = 2, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	5.914	**9.395**	**15.41**	**21.54**	**27.46**	**33.48**	**39.68**	**45.68**
δ-B-G	**5.827**	10.98	22.67	33.36	44.79	56.03	67.12	78.44
$\delta = 4, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	6.347	**9.717**	**15.75**	**21.87**	**27.84**	**33.94**	**40.01**	**46.03**
δ-B-G	**6.258**	11.76	24.02	35.79	47.41	59.89	71.23	83.55
$\delta = 4, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	8.135	12.67	**19.14**	**25.36**	**31.20**	**37.34**	**43.54**	**49.67**
δ-B-G	**6.259**	**11.99**	24.18	35.75	47.78	59.82	71.33	83.35

Experimental results with $\sigma = 120$

$\delta = 1, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.916**	**8.044**	**14.10**	**20.26**	**26.17**	**32.21**	**38.25**	**44.37**
δ-B-G	5.529	10.37	20.48	30.49	41.06	51.37	61.80	71.87
$\delta = 1, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.931**	**8.129**	**14.24**	**20.33**	**26.20**	**32.15**	**38.48**	**44.47**
δ-B-G	5.406	10.16	21.18	30.81	41.74	51.55	62.58	72.42
$\delta = 2, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.006**	**8.143**	**14.19**	**20.34**	**26.35**	**32.31**	**38.39**	**44.44**
δ-B-G	5.861	10.96	21.24	31.59	42.06	52.53	63.12	73.72
$\delta = 2, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.100**	**8.372**	**14.40**	**20.54**	**26.50**	**32.41**	**38.68**	**44.73**
δ-B-G	5.772	10.83	21.34	31.86	42.62	53.41	63.82	74.57
$\delta = 4, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.242**	**8.481**	**14.50**	**20.67**	**26.67**	**32.67**	**38.75**	**44.76**
δ-B-G	5.788	10.92	22.48	33.18	44.47	55.49	66.51	77.72
$\delta = 4, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.696**	**9.093**	**15.15**	**21.32**	**27.18**	**33.23**	**39.40**	**45.51**
δ-B-G	5.960	11.14	21.51	32.00	42.93	53.87	64.97	75.95

Experimental results with $\sigma = 180$

$\delta = 1, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.911**	**8.019**	**14.06**	**20.24**	**26.17**	**32.16**	**38.24**	**44.34**
δ-B-G	5.406	10.16	21.05	30.82	41.77	51.52	62.43	72.31
$\delta = 1, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.886**	**8.055**	**14.11**	**20.22**	**26.08**	**32.00**	**38.31**	**44.35**
δ-B-G	5.913	11.07	21.36	31.65	41.96	52.30	62.51	72.95
$\delta = 2, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.941**	**8.057**	**14.09**	**20.26**	**26.16**	**32.17**	**38.21**	**44.29**
δ-B-G	5.605	10.47	20.72	31.07	41.68	52.04	62.69	73.13
$\delta = 2, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.954**	**8.164**	**14.19**	**20.33**	**26.18**	**32.13**	**38.42**	**44.46**
δ-B-G	5.635	10.58	21.38	31.61	42.75	53.41	64.64	75.68
$\delta = 4, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.045**	**8.230**	**14.44**	**20.51**	**26.40**	**32.45**	**38.47**	**44.53**
δ-B-G	5.786	10.84	21.18	31.76	42.37	53.54	65.46	76.52
$\delta = 4, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.201**	**8.476**	**14.55**	**20.72**	**26.56**	**32.57**	**38.81**	**44.84**
δ-B-G	5.724	10.74	21.89	32.40	43.40	54.07	65.01	75.85

Experimental results on a Real Music problem with $\sigma = 101$

$\delta = 1, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.707**	**7.846**	**13.28**	**19.06**	**24.50**	**30.30**	**35.68**	**41.32**
δ-B-G	5.138	9.639	19.62	29.14	38.90	48.60	58.61	68.15
$\delta = 1, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**4.885**	**8.404**	**13.57**	**19.62**	**24.84**	**30.70**	**35.92**	**41.61**
δ-B-G	5.121	9.579	19.80	29.15	39.10	48.58	58.74	68.20
$\delta = 2, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.063**	**8.779**	**13.57**	**19.92**	**25.13**	**31.12**	**36.12**	**42.13**
δ-B-G	5.246	9.823	20.39	29.96	40.48	49.99	60.77	70.26
$\delta = 2, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.306**	**8.531**	**14.11**	**20.50**	**25.60**	**31.66**	**36.85**	**42.60**
δ-B-G	5.351	9.985	20.04	30.25	40.23	50.07	60.42	70.47
$\delta = 4, \alpha = 4$	10	20	40	60	80	100	120	140
(δ, α)-S-S	**5.610**	**9.077**	**14.59**	**21.10**	**26.31**	**32.26**	**37.41**	**43.32**
δ-B-G	5.677	10.49	21.26	31.68	42.39	52.75	63.52	73.72
$\delta = 4, \alpha = 8$	10	20	40	60	80	100	120	140
(δ, α)-S-S	5.864	**9.838**	**15.56**	**23.04**	**27.65**	**34.13**	**39.12**	**45.27**
δ-B-G	**5.581**	10.33	21.28	31.66	42.42	52.92	63.75	73.94

Experimental results show that most of the times our newly presented algorithm is faster than the one by Crochemore *et al.* Its superiority is more noticeable as the size of the pattern increases.

7 Conclusions

We have presented a new efficient $\mathcal{O}(mn)$-time algorithm for the δ-approximate string matching problem with α-bounded gaps. Extensive experimentation has shown that in most of the cases our algorithm is faster than the one by Crochemore *et al.*, which to our knowledge is the only solution present in literature for the same matching problem. The performances of our algorithm become more remarkable as the size of the pattern increases. In addition, our algorithm uses only $\mathcal{O}(m\alpha)$-space, rather than $\mathcal{O}(n)$-space, and it also computes the number of all distinct approximate matchings of the pattern at each position of the text.

We plan to reach a further speed-up of our algorithm by appropriate tuning. We also intend to generalize it to other variants of the approximate matching problem with gaps.

References

[CCF04] D. Cantone, S. Cristofaro, and S. Faro. Efficient algorithms for the δ-approximate string matching problem in musical sequences. pages 69–82, Czech Technical University, Prague, Czech Republic, 2004. Proc. of the Prague Stringology Conference '04.

[CCI+99] E. Cambouropoulos, M. Crochemore, C. S. Iliopoulos, L. Mouchard, and Y. J. Pinzon. Algorithms for computing approximate repetitions in musical sequences. In R. Raman and J. Simpson, editors, *Proceedings of the 10th Australasian Workshop On Combinatorial Algorithms*, pages 129–144, Perth, WA, Australia, 1999.

[CF03a] D. Cantone and S. Faro. Fast-Search: a new variant of the Boyer-Moore string matching algorithm. In J.D.P. Rolim (Eds.) M. Margraf, M. Mastrolli, editor, *LNCS 2647*, pages 47–58, 2003. Proc. of WEA 2003.

[CF03b] D. Cantone and S. Faro. Forward-Fast-Search: another fast variant of the Boyer-Moore string matching algorithm. pages 69–82, Czech Technical University, Prague, Czech Republic, 2003. Proc. of the Prague Stringology Conference '03.

[CIL+02] M. Crochemore, C. S. Iliopoulos, T. Lecroq, W. Plandowski, and W. Rytter. Three heuristics for δ-matching: δ-BM algorithms. In A. Apostolico and M. Takeda, editors, *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching*, number 2373 in Lecture Notes in Computer Science, pages 178–189, Fukuoka, Japan, 2002. Springer-Verlag, Berlin.

[CILP01] M. Crochemore, C. S. Iliopoulos, T. Lecroq, and Y. J. Pinzon. Approximate string matching in musical sequences. In M. Balík and M. Šimánek, editors, *Proceedings of the Prague Stringology Conference'01*, pages 26–36, Prague, Czech Republic, 2001. Annual Report DC–2001–06.

[CIM+02] M. Crochemore, C. Iliopoulos, C. Makris, W. Rytter, A. Tsakalidis, and K. Tsichlas. Approximate string matching with gaps, 2002.

[CIR98] T. Crawford, C. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11:71–100, 1998.

[KMGL88] S. Karlin, M. Morris, G. Ghandour, and M. Y. Leung. Efficient algorithms for molecular sequence analysis. *Proceedings of the National Academy of Science*, 85:841–845, 1988.

[KPR00] J. Karhumäki, W. Plandowski, and W. Rytter. Pattern-matching problems for two-dimensional images described by finite automata. *Nordic J. Comput.*, 7(1):1–13, 2000.